Recognizing P_3-structure: A Switching Approach*

RYAN B. HAYWARD†

Abstract. A P_3 is a set of three vertices of a graph that induces a chordless path; the P_3-structure of a graph is the set of all P_3's. Vašek Chvátal asked if there is a polynomial time algorithm to determine whether an arbitrary three-uniform hypergraph is the P_3-structure of some graph. The answer is yes: we describe such an algorithm.

Our algorithm is based on switching (as defined by Seidel), a characterization of graphs with the same P_3-structure, a sufficient condition for P_3-structure uniqueness, and a reduction of a certain partitioning problem to two-satisfiability.

Key Words. P_3-structure, P_4-structure, switching

1. The problem. A *hypergraph* $H = (V, E)$ is a set V of vertices together with a set E of subsets of vertices, called edges. A hypergraph is k*-uniform* if every edge has exactly k vertices. A 2-uniform hypergraph is usually referred to as a *graph*. In this paper we are particularly interested in graphs and 3-uniform hypergraphs. We shall refer to the edges of a 3-uniform hypergraph as *triples*.

A P_k of a graph is a subset of k of the vertices of the graph that induces a chordless path. For example, a P_3 is any subset $\{a, b, c\}$ of the vertex set such that exactly two of $\{a, b\}$, $\{a, c\}$, $\{b, c\}$ are edges of the graph. The P_k*-structure* of a graph G, written $P_k(G)$, is the set of all P_k's of G.

The main result of this paper is an algorithm that solves the P_3*-structure recognition problem*, posed by Vašek Chvátal [C]:

> *Find a polynomial time algorithm to recognize* P_3*-structures of graphs, that is, a polynomial time algorithm that given a 3-uniform hypergraph* H=(V,T)*, returns a graph* G *such that* P_3(G)=T*, or reports that no such graph exists.*

Another important result is a "switching approach" characterization of graphs with the same P_3-structure.

The P_3-structure recognition problem is interesting for at least two reasons. One, it can be considered a form of graph reconstruction. Two, it arises in the study of perfect graphs, as we now explain. (A graph is *perfect* if the clique size equals the chromatic number, for all induced subgraphs. See [BC] or [G] for an introduction to perfect graphs.)

---

* This research was supported by NSERC and the Alexander von Humboldt-Stiftung.

† Department of Mathematics and Computer Science, University of Lethbridge, 4401 University Drive, Lethbridge AB, Canada T1K 3M4; e-mail `hayward@cs.uleth.ca`

Chvátal conjectured and Bruce Reed proved that if two graphs have the same P_4-structure, then one is perfect if and only if the other is perfect [R]. This is usually referred to as the *Semi-Strong Perfect Graph Theorem*. It follows that to recognize perfect graphs in polynomial time, it is sufficient to recognize P_4-structures of perfect graphs in polynomial time. This observation motivated Chvátal to ask whether it is possible to recognize P_4-structures of arbitrary graphs (that is, not necessarily perfect graphs) in polynomial time, and at the same time to ask whether it is possible to recognize the P_3-structures of arbitrary graphs in polynomial time [C]. At present, there is no known polynomial algorithm for recognizing the P_4-structures of graphs, for recognizing the P_4-structures of perfect graphs, or for recognizing perfect graphs.

This paper is organized as follows. First, we discuss a naive algorithm for recognizing P_3-structure, and show why the algorithm is not polynomial. Next, we review known results in switching. Then, we discuss two characterizations, namely of graphs with the same P_3-structure, and of P_3-structures realizable by only one graph. Finally, we present our algorithm.

2. A naive algorithm. Suppose one is given a 3-uniform graph $H = (V, T)$ and asked to find a graph $G = (V, E)$ such that $P_3(G) = T$. Where might one start? An obvious idea is to try a naive inductive exhaustive search. Assume that at some stage one has found all graphs G' whose vertex set V' is a proper subset of V and such that $P_3(G') = T[V']$, the triples of T induced by V'. Now try to add some vertex to each possible G'. The main problem with this approach is that there may be many different graphs G' satisfying $P_3(G') = T[V']$ which do not extend to a graph G satisfying $P_3(G) = T$.

For instance, suppose $V = \{1, \ldots, n\}$ and

$$T = \{\, \{1,2,j\} | 3 \le j \le n-1 \} \,\cup\, \{\, \{1, n-1, n\}\, \}, \qquad \text{where } n \ge 5.$$

Now let $V' = V - \{n\}$, let A be any subset of $V' - \{1,2\}$, let $B = V' - \{1,2\} - A$, and let

$$\begin{aligned} E' \;=\; & \{\, \{1,2\}\, \} \;\cup\; \{\, \{1,a\} | a \in A\} \;\cup\; \{\, \{2,b\} | b \in B\} \\ & \cup\; \{\, \{x,y\} | x \in A,\ y \in A \quad \text{or} \quad x \in B,\ y \in B\}. \end{aligned}$$

It is a routine exercise to verify that the resulting graph $G' = (V', E')$ satisfies $P_3(G') = T[V']$. On the other hand, a straightforward induction leads to the conclusion that the only graph G satisfying $P_3(G) = T$ has edge set

$$\begin{aligned} E \;=\; & \{\, \{1,2\}, \{1,n-1\}, \{n-1,n\}\, \} \;\cup\; \{\, \{2,j\} | 3 \le j \le n-2\} \\ & \cup\; \{\, \{j,k\} | 3 \le j < k \le n-2\}. \end{aligned}$$

Thus we have an example where for any $n \geq 5$ there are (at least) 2^{n-3} different graphs with vertex set $V' = V - \{n\}$ that satisfy $P_3(G') = T[V']$, only one of which extends to a graph G such that $P_3(G) = T$.

This example raises two questions. First, are there "nice" starting sets of vertices, that is, initial sets of vertices V' for which the exhaustive search described above would end up considering only a relatively small number of graphs at each step? For instance, let V and T be as in the previous example and let $V' = \{1, 2, 3, n-1, n\}$. It is a routine exercise to verify that the only graph with P_3-structure $T[V']$ has edge set $\{\{1, 2\}, \{1, n-1\}, \{2, 3\}, \{n-1, n\}\}$. Now add vertices 4 through $n-2$ to V' one at a time; after each vertex is added there will be only one graph with P_3-structure $T[V']$. Thus, for the input example given, $V' = \{1, 2, 3, n-1, n\}$ turns out to be a "nice" starting set for exhaustive search. When do such sets exist, and how hard are they to find?

Second, is there some property of graphs with the same P_3-structure that makes keeping track of different graphs with the same P_3-structure unnecessary?

These questions motivate the design of our algorithm, and we shall return to them shortly. First we discuss the key notions of switching, as studied by J.J. Seidel.

3. Switching. We begin with some terminology. A *vertex partition* $[S_1, \ldots, S_k]$ of a graph is a family of pairwise non-intersecting sets whose union is the vertex set of the graph. Let $[A, B]$ be a vertex partition of a graph $G_1 = (V, E_1)$. With respect to this partition, replacing G_1 with $G_2 = (V, E_2)$, where E_2 is the symmetric difference of E_1 and $\{\{a, b\} | a \in A \text{ and } b \in B\}$, is called *switching*, or *switching across* $[A, B]$. Two graphs are *switching equivalent* if and only if one can be obtained from the other by switching. Switching has been studied by J.J. Seidel, and is often referred to as *Seidel switching.*

Switching is an equivalence relation; the associated equivalence classes are called *switching classes.* Graphs are switching equivalent if and only if their symmetric difference is a complete bipartite graph; the partition induced by the two parts of this bipartite graph is the vertex partition of the switching.

Analogous to P_k, define an I_k of a graph as a subset of k vertices that induces an independent set. In other words, an I_k is a set of k pairwise non-adjacent vertices of the graph. Analogous to P_3-structure, define the *IP_3-structure* of a graph to be the subsets of vertices that induce either I_3 or P_3. In other words, the IP_3-structure of a graph is the set of three-vertex subsets of the vertex set that induce an even number of edges. The IP_3-structure of a graph is a (not necessarily proper) superset of the P_3-structure of a graph. We call two graphs *IP_3-identical* if they have the same IP_3-structure. Restated in our terminology, the following result is equivalent to Lemma

3.9 and Corollary 3.5 in [S].

THEOREM 3.1. **[S]** *Two graphs are* IP_3*-identical if and only if they are switching equivalent.*

A *3-4-even hypergraph* is a 3-uniform hypergraph in which every set of four vertices induces an even number of triples. Seidel referred to such hypergraphs as *two-graphs*; as this term is used in another sense by some authors, we have chosen to introduce the mnemonically suggestive "3-4-even hypergraph". The following result is equivalent to Theorem 4.2 in [S].

THEOREM 3.2. **[S]** *The* IP_3*-structure of any graph is a 3-4-even hypergraph. Furthermore, given a 3-4-even hypergraph* H=(V,T)*, there exists a non-empty switching class with vertex set* V*, such that the* IP_3*-structure of (every graph in) the switching class is* T.

Thus there is a one-to-one correspondence between switching classes of graphs and IP_3-structures, and between IP_3-structures and 3-4-even hypergraphs. The former correspondence is obtained by taking the IP_3-structure of any graph in the class; the latter correspondence is just equality.

4. Nice starting sets and P_3-identical graphs. Near the end of §2 it was pointed out that a subset of the vertex set of a 3-uniform hypergraph is a nice starting set for the naive P_3-structure recognition algorithm if, for that subset of vertices, there is exactly one solution graph.

Call a 3-uniform hypergraph *realizable* (respectively *uniquely realizable*) if there is some (respectively exactly one) graph with that hypergraph as its P_3-structure; otherwise, it is *non-realizable.* Call a graph whose P_3-structure is uniquely realizable P_3*-unique.* Using this terminology, the observation of the previous paragraph is that vertex subsets that induce uniquely realizable hypergraphs make nice starting sets for the naive recognition algorithm. This raises the question: Which hypergraphs are uniquely realizable?

We do not answer this question in general. For small vertex sets, the question can be answered by enumerating all possible graphs. In particular, such enumeration reveals that (ignoring the trivial 3-uniform hypergraph consisting of one vertex and no triples) there are, up to isomorphism, three uniquely realizable 3-uniform hypergraphs on at most five vertices, namely

$$\{\,\{1,2,3\},\{1,2,4\},\{1,3,4\}\,\},$$
$$\{\,\{1,2,3\},\{2,3,4\},\{3,4,5\}\,\},$$
$$\{\,\{1,2,3\},\{1,4,5\},\{2,3,4\},\{2,4,5\}\,\}.$$

The corresponding P_3-unique graphs have respective edge sets

$$\{\ \{1,2\},\{1,3\},\{1,4\}\ \},$$
$$\{\ \{1,2\},\{2,3\},\{3,4\},\{4,5\}\ \},$$
$$\{\ \{1,2\},\{1,4\},\{2,3\},\{2,4\},\{4,5\}\ \}.$$

We shall refer to these three graphs, or isomorphic copies, as the *claw*, P_5, and the *bull*, respectively. A connected graph containing at least one of these three graphs has the following useful characterization.

LEMMA 4.1. *Let* G *be a connected graph. Then* G *contains at least one of the claw,* P_5*, or the bull if and only if* G *contains* I_3.

Proof. ($\Longrightarrow$) The claw, P_5 and the bull each contain I_3.

($\Longleftarrow$) Argue by induction. Let v be any vertex of G such that $G' = G - v$ is connected. If G' contains I_3, we are done by hypothesis, so suppose not. Let $\{v,x,y\}$ be an I_3 of G, and let A,B,C,D be the sets of neighbors of v that are adjacent respectively to only x, only y, both x and y, and neither x nor y. Since G' is I_3-free, D is empty. If C is non-empty, G contains a claw; if both and A and B are non-empty, G contains P_5 or the bull. Suppose then that C and one of A or B (say B) are empty. Let P be a shortest path from y to a vertex of A. If P has more than three vertices, $G[P \cup \{v\}]$ contains P_5; if P has only three vertices, $G[P \cup \{v,x\}]$ contains the bull or a claw. □

The above lemma suggests the beginnings of our algorithm. If a 3-uniform hypergraph is realizable by a connected graph that contains I_3, then the hypergraph contains as an induced hypergraph the P_3-structure of at least one of the claw, P_5, or the bull, any of which can be used as a nice starter for the naive algorithm. On the other hand, if a hypergraph is realizable only by I_3-free graphs, then the following theorem applies. We shall see later how to proceed in this case.

THEOREM 4.2. *Let G be a connected graph. Then G is I_3-free if and only if* $\mathrm{P}_3(G)$ *is a 3-4-even hypergraph.*

Proof. ($\Longleftarrow$) Suppose that G contains I_3. By lemma 4.1, G contains at least one of the claw, P_5, or the bull, and each of these graphs has a set of four vertices that induces an odd number of P_3's. Thus $P_3(G)$ is not a 3-4-even hypergraph.

($\Longrightarrow$) Suppose that $P_3(G)$ is not a 3-4-even hypergraph. Then some set $S = \{w,x,y,z\}$ of vertices of G induces either one or three P_3's in G. In the former case, $G[S]$ is isomorphic to the union of an isolated vertex and a P_3; in the latter case, $G[S]$ is isomorphic to the claw. In each case, G contains an I_3. □

We now address the question of determining when two graphs are P_3-*identical*, that is, have the same P_3-structure. Somewhat surprisingly, such graphs are necessarily switching equivalent; we shall prove this shortly. The converse does not hold, since switching may interchange P_3's and I_3's. Such an interchange occurs if and only if some vertex in one part of the switching partition is adjacent to both or neither of a non-adjacent pair of vertices in the other part. By forbidding such three-vertex configurations, we obtain a restricted form of switching which preserves P_3-structure.

Thus, define a P_3-*partition* of a graph as a vertex partition consisting of two parts, such that for each of the two parts, each vertex in that part is adjacent to exactly one vertex of each non-adjacent pair of vertices of the other part. Observe that for any graph $G = (V, E)$, the trivial partition $[V, \phi]$ is always a P_3-partition. Define P_3-*switching* as switching across a P_3-partition. Although it is not immediately obvious, P_3-switching is an equivalence relation; we call two graphs P_3-*switching equivalent* if one can be obtained from the other by P_3-switching.

THEOREM 4.3. *Two connected graphs are* P_3-*identical if and only if they are* P_3-*switching equivalent.*

Proof. ($\Longleftarrow$) Since switching preserves IP_3-structure, to show that P_3-switching preserves P_3-structure, it is enough to show that no P_3's are switched into I_3's or vice versa. Since P_3-switching is an equivalence relation and in particular symmetric, the "vice versa" in the previous statement may be omitted. So consider a P_3-partition of a graph G_1, and let G_2 be the graph obtained by switching G_1 across this partition. Let S be a P_3 of G_1. If S is situated wholly within one of the parts, then $G_1[S]$ is unaffected by the switch, so $G_1[S] = G_2[S]$ and thus $G_2[S]$ is a P_3. If S is situated with one end vertex of $G_1[S]$ in one part and the other two vertices in the other part, then $G_2[S]$ is again a P_3, although the edge set is different from that of $G_1[S]$. Finally, S cannot be situated with the middle vertex of $G_1[S]$ in one part and the other two vertices in the other part, by the definition of P_3-partition.

($\Longrightarrow$) Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be connected graphs such that $P_3(G_1) = P_3(G_2)$. First suppose that neither G_1 nor G_2 contains I_3. Thus $IP_3(G_1) = P_3(G_1) = P_3(G_2) = IP_3(G_2)$. By Theorem 3.1, G_1 and G_2 are switching equivalent, namely there is a switching between G_1 and G_2 that preserves their IP_3-structure. However, since both G_1 and G_2 are I_3-free, this switching preserves their P_3-structure, and so is a P_3-switching. (Otherwise, one vertex in one part is adjacent to both or neither of two non-adjacent vertices in the other part. But then these three vertices induce I_3 in exactly one of G_1 and G_2, contradicting the assumption that both are I_3-free). Thus G_1 and G_2 are P_3-switching equivalent.

Next suppose that one of G_1 or G_2, say G_1, contains I_3. To finish the proof, we argue that G_1 is P_3-unique, from which it follows that G_1 and G_2 are identical and so trivially P_3-switching equivalent. We wish to show that $E_1 = E_2$. Let S be any smallest subset of V that induces one of the claw, the bull, or P_5. (S exists by Lemma 4.1. S has size four if G_1 contains the claw, and size five otherwise). Argue by induction on $|V - S|$.

Basis case. $|V - S| = 0$. Thus $S = V$, and G_1 is one of the claw, the bull, or P_5. An enumeration of all graphs on four or five vertices confirms that G_1 is P_3-unique.

Inductive case. $|V - S| > 0$. Let v_* be any vertex of $V - S$ such that $G_1 - v_*$ is connected. (Such a vertex exists; for example, let v_* be any vertex whose distance to a vertex of S is maximized, over all vertices of $V - S$). Since S is a subset of $G_1 - v_*$, $G_1 - v_*$ contains I_3, and by inductive hypothesis is P_3-unique. Also, $P_3(G_1) = P_3(G_2)$, so $P_3(G_1 - v_*) = P_3(G_2 - v_*)$. Thus $G_1 - v_* = G_2 - v_*$. It remains only to prove that $N_1(v_*) = N_2(v_*)$, where $N_j(v)$ represents the neighborhood of v in the graph G_j.

Label the vertices of V so that $\{v_1, v_2, v_3\}$ is the I_3 of $G_1[S]$. We claim that for some $v_t \in \{v_1, v_2, v_3\}$,

$$\{v_*, v_t\} \in E_1 \iff \{v_*, v_t\} \in E_2 \ .$$

To prove the claim, first suppose that $\{v_*, x, y\}$ is a P_3 of G_1 (and so also G_2), for some $\{x, y\}$ a subset of $\{v_1, v_2, v_3\}$. This and the fact that $\{x, y\}$ is not an edge of $G_1 - v_* = G_2 - v_*$ implies that both $\{v_*, x\}$ and $\{v_*, y\}$ are edges of G_1 (and so G_2), and the claim holds for $v_t = x$.

Next suppose that $\{v_*, x, y\}$ is not a P_3 of G_1 (and so not of G_2), for each $\{x, y\}$ a subset of $\{v_1, v_2, v_3\}$. It follows that at most one of $\{v_*, v_1\}, \{v_*, v_2\}, \{v_*, v_3\}$ is an edge of G_1, and at most one is an edge of G_2. Thus at least one of these three sets is an edge of neither G_1 nor G_2, and the proof of the claim is complete.

Now relabel $V - v_*$ as $\{v_0, v_1, v_2, \ldots, v_k\}$ so that $v_t = v_0$, and for each $j \geq 1$,

$$\text{for some } \ i < j, \ \{v_i, v_j\} \in E_1[V - v_*] \ \ (= E_2[V - v_*]) \ .$$

Such a labelling exists, since $G_1 - v_*$ $(= G_2 - v_*)$ is connected. Since $\{v_0, v_1\}$ is an edge of $G_1 - v_*$, we have

$$\{v_1, v_*\} \in E_1 \iff (\ \{v_0, v_*\} \in E_1 \ ) \ \triangledown \ (\ \{v_*, v_0, v_1\} \in P_3(G_1) \ )$$

where here $\triangledown$ stands for the "exclusive or" operation. Since the corresponding property holds for E_2 and G_2 in place of E_1 and G_1, and since by the claim $\{v_*, v_0\}$ is in E_1 if and only if $\{v_*, v_0\}$ is in E_2, it follows that $\{v_*, v_1\}$ is in E_1 if and only if $\{v_*, v_1\}$ is in E_2. Continuing this argument for v_2, v_3, and so on, it follows that $N_1(v_*) = N_2(v_*)$. Thus $E_1 = E_2$ and the theorem is proved. □

Before describing the initial phase of our algorithm, we discuss some relevant aspects of connectivity. A hypergraph is *connected* if for any pair of vertices a and z

there is a finite sequence of edges $(e_1, \ldots, e_j)$ such that a is in e_1, z is in e_j, and each consecutive pair of edges in the sequence intersects in at least one vertex. To solve the P_3-structure recognition problem (that is, to determine whether a given 3-uniform hypergraph is realizable), it is sufficient to solve the problem independently for each of the connected components of the input hypergraph, for the following reason. Suppose that the input 3-uniform hypergraph $H = (V, T)$ is not connected, and that for each 3-uniform hypergraph $H' = (V', T')$ corresponding to a connected component of H there is a graph $G' = (V', E')$ such that $P_3(G') = T'$. Then the graph $G = (V, E)$ whose edge set E is the union of the edge sets E' satisfies $P_3(G) = T$. On the other hand, if for any subset V^* of vertices and corresponding set $T[V^*]$ of triples there exists no graph G^* such that $P_3(G^*) = T[V^*]$, then there exists no graph G such that $P_3(G) = T$.

Thus, as a corollary to Theorem 4.2 and the last part of the proof of Theorem 4.3, we have a useful partial answer to the question of which hypergraphs are uniquely realizable and make nice starting sets.

COROLLARY 4.4. *Let* H *be a connected 3-uniform hypergraph, and suppose that* H *is realizable by a graph* G*. Then* G *is* I_3*-free if* H *is 3-4-even, and* G *is unique and contains at least one of the claw,* P_5*, or the bull if* H *is not 3-4-even.*

The initial phase of our algorithm consists of checking whether the input hypergraph is 3-4-even, and if so, finding a graph of the associated switching class. We refer to these two tasks as *Test 3-4-even* and *Represent 3-4-even* respectively. Test 3-4-even can be implemented by simply checking each four-element subset of the vertex set. As pointed out by Seidel [S], Represent 3-4-even can be implemented by constructing a desired graph F in the following manner:

> *Select any vertex* v *of the vertex set* V*, and make* v *adjacent to all vertices of* V−v*. For each pair of vertices of* G−v *make the pair adjacent if and only if the pair together with* v *is not a triple of* H.

Verifying that Represent 3-4-even is correct, namely that $P_3(F) = H$, amounts to verifying that a three-element subset X of V is a triple of H if and only if it induces an even number of edges of F. If X contains v, this is immediate from the construction; if X does not contain v, this follows from the fact that H is 3-4-even, by considering $X \cup \{v\}$.

5. The algorithm. In this section we present the algorithm. We begin with two lemmas which motivate different parts of the algorithm.

LEMMA 5.1. *Let* H=(V,T) *be a connected 3-4-even hypergraph and let* F *be a graph such that* IP_3(F)= T*. Then a graph* G *satisfies* P_3(G)=T *if and only if* F *and*

G *are switching equivalent and* G *is* I_3*-free.*

Proof. Let H and F be as stated in the theorem. Suppose that some graph G satisfies $P_3(G) = T$. Since H is 3-4-even, Corollary 4.4 implies that G is I_3-free, so $IP_3(G) = P_3(G) = T = IP_3(F)$, so F and G are switching equivalent. Next suppose that a graph G is I_3-free and switching equivalent to F. Then $P_3(G) = IP_3(G) = IP_3(F) = T$, so $P_3(G) = T$. □

LEMMA 5.2. *Let* G=(V,E) *be a connected graph with a proper subset* V′ *of* V *such that* G[V′] *is connected and not a clique. Then some* P_3 *of* G *contains exactly one vertex of* V−V′*.*

Proof. In a graph, a vertex is the middle vertex of a chordless path with three vertices if and only if its neighborhood is not a clique, and the end vertex of such a path if and only if one of its neighbors is adjacent to one of its non-neighbors. It follows that in a graph which is connected and not a clique, every vertex is in a P_3.

Let G, V and V' be as stated in the lemma. Since G and $G[V']$ are connected, there is some vertex v in $V - V'$ such that $G^* = G[V' \cup \{v\}]$ is connected. Since $G[V']$ is not a clique, G^* is not a clique. By the preceding argument, v is in some P_3 of G^*, and this P_3 contains exactly one vertex of $V - V'$. □

A *nice starter* of a 3-uniform hypergraph $H = (V, T)$ is a subset S of the vertices, such that $H[S]$ is the P_3-structure of one of the claw, P_5, or the bull. An *IP-partition* of the vertices of a graph $G = (V, E)$ is a partition $[A, B]$ of V such that

> *for each pair of non-adjacent vertices in* A *(respectively* B*), every other vertex in* A *(respectively* B*) is adjacent to at least one of the pair, and every vertex in* B *(respectively* A*) is adjacent to at most one of the pair.*

An equivalent definition is that G has no I_3 contained entirely in one of the parts of the partition, and no P_3 both of whose edges cross the partition.

We now present the algorithm. As mentioned at the end of the previous section, the first step is to determine whether the input hypergraph H is 3-4-even, by performing Test 3-4-even. If H is not 3-4-even, the next step is to search for a nice starter, since any realizing graph must be unique and contain one of the claw, P_5 or the bull. A realization of the starter can be extended iteratively into a realization of the whole hypergraph, by adding one vertex at a time. These tasks constitute Algorithm Ia.

If H is 3-4-even, the next step is to find a graph F of the associated switching class, by performing Represent 3-4-even. By Lemma 5.1, the hypergraph is realizable if and only if F can be switched into an I_3-free graph G, in which case G is a realization. We shall see that such a switching exists if and only if F admits an IP-partition, and that determining whether F admits an IP-partition can be formulated as a 2-satisfiability problem. This formulation together with the solution of the resulting 2-satisfiability problem constitutes Algorithm Ib.

Algorithm I. **P_3-structure recognition.**
Input. A connected 3-uniform hypergraph $H = (V, T)$.
Output. Either a graph G such that $P_3(G) = T$, or the message 'non-realizable'.

1. Perform Test 3-4-even, then Step 2 if H is 3-4-even, and Step 3 if H is not.
2. Perform Represent 3-4-even, and with the resulting graph F, perform Algorithm Ib.
3. Perform Algorithm Ia.

Algorithm Ia. **Nice Starter Search and Continuation.**
Input. A connected 3-uniform hypergraph $H = (V, T)$ that is not 3-4-even.
Output. Either a graph G such that $P_3(G) = T$, or 'non-realizable'.

1. Search for a nice starter S in H.
   If none is found, return 'non-realizable'.
2. $r \leftarrow |S|$, $V_r \leftarrow S$, $E_r \leftarrow$ edges so that $P_3(G_r = (V_r, E_r)) = T[V_r]$, and label V_r as $\{v_1, v_2, \ldots, v_r\}$ so that for each j with $2 \leq j \leq r$, for some $i < j$, $\{v_i, v_j\} \in E_r$.
3. For $k \leftarrow r + 1$ to n do

3.1. Search for a vertex v_* of $V - V_{k-1}$ such that some triple of T contains v_* and two vertices of V_{k-1};
if found then $v_k \leftarrow v_*$; if not return 'non-realizable',

3.2. $V_k \leftarrow V_{k-1} \cup \{v_k\}$, $N_k^1 \leftarrow \{\{v_1, v_k\}\}$,

3.3. for $j \leftarrow 2$ to $k - 1$ do
- select any $i < j$ so that $\{v_i, v_j\} \in E_{k-1}$,
- add $\{v_j, v_k\}$ to N_k^1 if and only if
  ($\{v_i, v_k\} \notin N_k^1$ and $\{v_i, v_j, v_k\} \in T$) or
  ($\{v_i, v_k\} \in N_k^1$ and $\{v_i, v_j, v_k\} \notin T$),

```
3.4.        N_k^2 ← V_{k-1} − N_k^1 ,
3.5.        if      P_3((V_k, E_{k-1} ∪ N_k^1)) = T[V_k] then E_k ← E_{k-1} ∪ N_k^1
            else if P_3((V_k, E_{k-1} ∪ N_k^2)) = T[V_k] then E_k ← E_{k-1} ∪ N_k^2
            else return 'non-realizable'.
4.     Return G = (V, E_n).
```

Algorithm Ib. Switching to I_3-free.

Input. A connected 3-uniform hypergraph $H = (V, T)$ that is 3-4-even, and a graph $F = (V, E)$ such that $IP_3(F) = T$.

Output. Either a graph G such that $P_3(G) = T$, or 'non-realizable'.

```
1.     Search for an I_3 {v_i, v_j, v_k} of F.  If none is found,
       then G ← F and return G.
2.     Construct a 2-satisfiability formula C as follows.
2.1.        C ← φ.  Create a variable a_x for each v_x in V.
2.2.        S_A ← set containing v_i, v_j, and all vertices of
            V − {v_i, v_j} adjacent to both v_i and v_j.  For each vertex
            v_t in S_A, add clause (a_t) to C .
2.3.        S_B ← vertices of V − {v_i, v_j} adjacent to neither
            v_i nor v_j.  For each v_u in S_B, add clause (ā_u) to C .
2.4.        For each I_3 {v_x, v_y, v_z} of F do
                 if v_x ∈ S_A then add (ā_y ∨ ā_z) to C ,
                 if v_y ∈ S_A then add (ā_x ∨ ā_z) to C ,
                 if v_z ∈ S_A then add (ā_x ∨ ā_y) to C ,
                 if v_x ∈ S_B then add (a_y ∨ a_z) to C ,
                 if v_y ∈ S_B then add (a_x ∨ a_z) to C ,
                 if v_z ∈ S_B then add (a_x ∨ a_y) to C .
2.5         For each P_3 {v_x, v_y, v_z} of F such that {v_x, v_y} ∈ E(F) and
            {v_y, v_z} ∈ E(F) do
                 if v_x ∈ S_A then add (a_y ∨ ā_z) to C ,
                 if v_x ∈ S_B then add (ā_y ∨ a_z) to C ,
                 if v_y ∈ S_A then add (a_x ∨ a_z) to C ,
                 if v_y ∈ S_B then add (ā_x ∨ ā_z) to C ,
                 if v_z ∈ S_A then add (a_y ∨ ā_x) to C ,
                 if v_z ∈ S_B then add (ā_y ∨ a_x) to C .
3.     Repeat 2, first with v_j, v_k, v_i replacing v_i, v_j, v_k respectively,
       and then v_k, v_i, v_j replacing v_i, v_j, v_k respectively.  Let C' and C''
       be the corresponding 2-satisfiability clauses created.
```

```
4.  Determine whether there is a solution to any of 𝒞 , 𝒞′ , or 𝒞″ .
    If so, A ← vertices v_j such that a_j is true in the solution, and
    return the graph G obtained by switching F across [A, V − A].
    If not, return 'non-realizable'.
```

THEOREM 5.3. *Algorithm I is correct.*

Proof of theorem. Let $H = (V, T)$ be an input hypergraph for the algorithm. First suppose that H is not 3-4-even. We wish to show that Algorithm Ia is correct. Observe that if Algorithm Ia returns a graph G (as opposed to the message 'non-realizable'), then Step 4 was reached, so Step 3.5 completed with $k = n$, so $P_3(G) = T[V_n] = T$ was verified, so G realizes H. Thus to show the correctness of Algorithm Ia, we need only show that Algorithm Ia reaches Step 4 if H is realizable.

Suppose then that H is realizable, say by a graph G^*. By Corollary 4.4, G^* is unique and contains the claw, P_5, or the bull. Thus H contains a nice starter, so one will be found in Step 1, and Algorithm Ia reaches Step 2. Step 2 constructs a graph realizing the nice starter S found in Step 1, and orders the vertices so that each vertex is adjacent to at least one previous vertex in the order. Such an ordering exists for any connected graph, so Algorithm Ia reaches Step 3.

We wish to show that Algorithm Ia completes Step 3 for each value of k, so suppose Step 3 is about to execute with $k = t$. We may assume that $P_3(G_{t-1} = (V_{t-1}, E_{t-1})) = T[V_{t-1}]$, either because $t = r + 1$ and Step 2 completed, or because $t > r+1$ and Step 3.5 was reached with $k = t-1$. We also have $P_3(G^*[V_{t-1}]) = T[V_{t-1}]$ by our definition of G^*. This and the fact that $H[V_{t-1}]$ is uniquely realizable (since it is connected and contains the nice starter S) implies that $G_{t-1} = G^*[V_{t-1}]$.

Now observe that G^* is connected (since H is connected), $G^*[V_{t-1}]$ is connected (since by our construction $H[V_{t-1}]$ is connected), and $G^*[V_{t-1}]$ is not a clique (since S is a subset of V_{t-1} and $G^*[S]$ is one of the claw, P_5 or the bull). Thus Lemma 5.2 implies that some vertex of $V - V_{t-1}$ is in some P_3 of G^* with two vertices of V_{t-1}, and so in some triple of T with two vertices of V_{t-1}. Thus the vertex described in Step 3.1 will be found, so Step 3.1 completes with $k = t$.

Steps 3.2, 3.3, and 3.4 complete. The selection described in Step 3.3 is possible, since the graph $G_{t-1}[V_{j-1}]$ is connected for $1 \le j \le t-1$.

Now by induction on j, with $2 \le j \le t-1$, it is routine to verify that the neighborhood of v_t in $G^*[V_t]$ is exactly N_t^1 if $\{v_1, v_k\}$ is an edge of G^*, and N_t^2 if it is not. Thus $G^*[V_t] = (V_t, E_{t-1} \cup N_t^1)$ or $(V_t, E_{t-1} \cup N_t^2)$, and one of the two P_3-structure checks of Step 3.5 will be successful. Thus Step 3.5 completes with $k = t$, Algorithm

Ia is correct, and the proof of the theorem is complete if H is not 3-4-even.

Suppose then that H is 3-4-even. By Theorem 3.2, Represent 3-4-even returns a graph F satisfying $IP_3(F) = T$. To complete the proof, we need only show Algorithm Ib is correct. By Lemma 5.1, we need only show that any graph G returned by Algorithm Ib is I_3-free and switching equivalent to F, and that if H is realizable, then some graph is returned by Algorithm Ib.

Consider the latter first: suppose that H is realizable. If F is I_3-free, the algorithm returns a graph (namely $G = F$), so assume F contains I_3. By Lemma 5.1, there is some I_3-free graph G^* such that F and G^* are switching equivalent. Let $[A^*, B^*]$ be the associated *IP*-partition of F. We wish to show Algorithm Ib finds some *IP*-partition of F.

Since F contains an I_3, the algorithm finds vertices $\{v_i, v_j, v_k\}$. By exchanging the names of A^* and B^* if necessary, we may assume that A^* contains at least two of $\{v_i, v_j, v_k\}$. It is a routine exercise to verify that the assignment of truth values to the boolean variables a_j to match the partition A^* will satisfy $\mathcal{C}$, $\mathcal{C}'$, $\mathcal{C}''$ respectively in the case where A^* contains both of $\{v_i, v_j\}$, $\{v_j, v_k\}$ or $\{v_i, v_k\}$ respectively. Thus there is at least one solution to at least one of $\mathcal{C}$, $\mathcal{C}'$, $\mathcal{C}''$ and so the algorithm returns a partition.

Thus to complete the proof, we need only show that any graph G returned by Algorithm Ib is I_3-free and switching equivalent to F. This holds if F contains no I_3 (since the Algorithm returns $G = F$ in this case), so assume F contains some I_3. Let $[A, B]$ be the partition returned by Algorithm Ib; we wish to show that $[A, B]$ is an *IP*-partition of F. By relabelling if necessary the vertices of $\{v_i, v_j, v_k\}$ (the I_3 of F selected in Step 1), we may assume that the $[A, B]$ partition corresponds to a solution of $\mathcal{C}$ (as opposed to $\mathcal{C}'$ or $\mathcal{C}''$). Thus S_A is the union of v_i and v_j and all vertices of $V - \{v_i, v_j\}$ that are adjacent to both of $\{v_i, v_j\}$, and S_B is the set of all vertices of $V - \{v_i, v_j\}$ that are adjacent to neither of $\{v_i, v_j\}$. There are two cases to consider.

Case 1. $I = \{v_p, v_q, v_r\}$ is an I_3 of F.

We wish to show that I is not contained in A or in B. If $S_A \cap I$ or $S_B \cap I$ is nonempty, then the desired conclusion follows from that fact that $\mathcal{C}$ contains a clause created by Step 2.4, corresponding to $\{v_x, v_y, v_z\} = I$. So assume that $S_A \cap I = \phi$ and $S_B \cap I = \phi$. In particular, neither v_i nor v_j is in I, and each vertex of I is adjacent to exactly one vertex of $\{v_i, v_j\}$. Thus two vertices of I (by relabelling if necessary, say v_p,v_q) are both adjacent to one vertex of $\{v_i, v_j\}$ and not the other. Because v_p and v_q are each adjacent to a vertex of S_A, the three vertices form a P_3 of F, and by the appropriate case of Step 2.5, v_p and v_q are not both placed in B. On the other hand, because v_p and v_q are each nonadjacent to a vertex of S_A, these three vertices form an I_3 of F, and so by the appropriate case of Step 2.4, v_p and v_q are not both placed in A. It follows that I is not contained in A or in B.

Case 2. $P = \{v_p, v_q, v_r\}$ is a P_3 of F, with edges $\{v_p, v_q\}$ and $\{v_q, v_r\}$. We wish to show that the vertices of P are placed so that at most one of the edges of $F[P]$ crosses the $[A, B]$ partition. As in Case 1, we may assume that $S_A \cap P = \phi$ and $S_B \cap P = \phi$. Thus neither v_i nor v_j is in P, and each vertex of P is adjacent to exactly one vertex of $\{v_i, v_j\}$. By relabelling v_i and v_j if necessary, we may assume that v_i is adjacent to at least two vertices of P. If v_i is adjacent to both v_p and v_r, then by an argument similar to the concluding argument of Case 1, we conclude that v_p and v_r are placed neither both in A nor both in B, and we are done.

Suppose then that v_i is not adjacent to both v_p and v_r. Since v_i is adjacent to at least two of the vertices of P, by relabelling v_p and v_r if necessary, we may assume that v_i is adjacent to v_p and v_q and not v_r. Thus v_j is adjacent to v_r and neither v_p nor v_q. Because $\{v_i, v_q, v_r\}$ is a P_3 of F, the appropriate case of Step 2.5 forbids the placement of v_r and v_q with v_q in B and v_r in A. Similarly, because $\{v_j, v_q, v_r\}$ is a P_3 of F, the appropriate case of Step 2.5 forbids the placement of v_r and v_q with v_q in A and v_r in B. It follows that v_q and v_r are placed either both in A or both in B, and at most one edge of $F[P]$ crosses the $[A, B]$ partition. This completes the proof of the theorem. □

THEOREM 5.4. *Algorithm I runs on hypergraph $H = (V, T)$ in time polynomial in $|V|$ and $|T|$.* □

The proof of this theorem follows immediately from the polynomiality of the various parts of the algorithm, namely Test 3-4-even, Represent 3-4-even, Algorithm Ia and Algorithm Ib. Since a 2-satisfiability formula can be solved in time proportional to the number of clauses plus literals (see for example [EIS]), a straightforward implementation of our algorithm runs in time $O(|V|^5)$. An implementation of a more complicated version of this algorithm runs in time $O(|V|^3)$ [H2].

6. Conclusions. We have combined a switching-based characterization of P_3-identical graphs, a sufficient condition for P_3-uniqueness, and a 2-satisfiability reduction to yield a "yes" answer to Chvátal's question "can P_3-structure be recognized in polynomial time?".

This question can be asked with any fixed graph F in place of P_3. If F is complete, the answer is "yes". Also, for any graph F the answer is the same as for the complement of F. Thus the answer is "yes" for all graphs F with at most three vertices. For all graphs F with four or more vertices, the question is open. We close by asking whether there is any graph F for which the answer is not "yes", namely, for which recognizing F-structure is NP-hard.

Postscript. Upon presenting these results at the *DIMACS June 1993 Workshop on Perfect Graphs at Princeton*, the author learned of several independent results related to this work. Kratochvíl, Nešetřil, and Zýka investigated the complexity of switching an arbitrary input graph to a graph possessing a given property P [KNZ]. In particular, they showed that switching to a P_3-free graph is polynomial. In an unpublished manuscript, they further showed that switching to a triangle-free graph is polynomial; this result, equivalent to showing that switching to an I_3-free graph is polynomial, follows from our Algorithm Ib and Theorem 5.4. On the other hand, Colbourn and Corneil showed that determining whether two *fixed* graphs are switching equivalent is polynomial time equivalent to deciding graph isomorphism [CC]. Simonyi characterized a class of 3-uniform hypergraphs with certain entropy splitting properties; in the process, the notion of IP_3 structure arose [Si]. Ding has a polynomial time algorithm for recognizing the P_4-structures of trees [D].

Acknowledgements. I thank Vašek Chvátal for informing me of this problem, and François Jaeger for telling me of Seidel switching. I also thank Michel Burlet, Jean Fonlupt, Andràs Frank, Mike Hallett, Stefan Hougardy, Bruce Reed, Mike Saks and Andràs Sebö for conversations and comments about this work, and the referees for their constructive criticisms.

The original P_3-structure recognition algorithm [H1] does not use Seidel switching. A preliminary version of this paper [H2] includes algorithms which are more complicated but take only $O(|V|^3)$ time.

REFERENCES

[BC] C. Berge and V. Chvátal (Eds.), "Topics on Perfect Graphs", *Annals of Discrete Math.* **21**, North-Holland, Amsterdam 1984.

[C] V. Chvátal, private communication, 1987.

[CC] C. Colbourn, D. Corneil, On deciding switching equivalence of graphs, *Discrete Applied Mathematics* **2** (1980) 181-184.

[D] G. Ding, Recognizing the P_4-structure of a tree, manuscript, January 1993.

[EIS] S. Even, A. Itai and A. Shamir, On the complexity of timetable and multi-commodity flow problems, *SIAM J. Comput.* **5** (1976) 691-703.

[G] M. Golumbic, "Algorithmic graph theory and perfect graphs", Academic Press, San Diego, 1980.

[H1] R. Hayward, Recognizing P_3-structure, Forschungsintstitut für Diskrete Mathematik Report 90625-OR, Rheinische Friedrich-Wilhelms-Universität Bonn, January 1990.

[H2] R. Hayward, P_3-structure Recognition via Switching: The Algorithms, University of Lethbridge DMCS TR-CS-05-93.

[KNZ] J. Kratochvíl, J. Nešetřil, and O. Zýka, On the computational complexity of Seidel's switching, *Proc. 4th Czechoslavakian Symposium on Combinatorics*, in "Graphs and Complexity", J. Nešetril and M. Fielder (eds), Elsevier Science Publ. B.V. 1992.

[R] B. Reed, A semi-strong perfect graph theorem, *J. Comb. Theory B* **43** (1987) 223-240.

[S] J.J. Seidel, A survey of two-graphs, *Proc. Intern. Colloqu. Teorie Combinatoire* (Rome 1973), Tomo I, Accad. Naz. Lincei (1976) 481-511.

[Si] G. Simonyi, Entropy spliting hypergraphs, manuscript, June 1993, submitted to *J. Comb. Theory B.*

Suggested Running Head: Recognizing P_3-structure

List of Symbols:

ABCDEFGHIJKLMNOPQRSTUVWXYZ-
abcdefghijklmnopqrstuvwxyz 0123456789 P$_3$, ; : . ? ( ) ‘ ’ [ ]

ABCDEFGHIJKLMNOPQRSTUVWXYZ-
abcdefghijklmnopqrstuvwxyz 0123456789 P$_3$, ; : . ? ( ) ‘ ’

ABCDEFGHIJKLMNOPQRSTUVWXYZ-
abcdefghijklmnopqrstuvwxyz 0123456789 P_3 *, ; : . ? ( ) ‘ ’*

`ABCDEFGHIJKLMNOPQRSTUVWXYZ-`
`abcdefghijklmnopqrstuvwxyz 0123456789 ()‘’ , ; : . ? ( ) ‘ ’`

$\Longleftarrow \Longrightarrow \Longleftrightarrow \leftarrow \bullet$
$\in = \geq \leq < >\ ( ) \{ \} [ ] \,' \,'' \,^* + - \cup \cap | \phi \overline{a} \vee$
$v_1\ v_a\ v_n\ v_{k-1}$

$\mathcal{C}\ \mathcal{C}'\ \mathcal{C}''$

Recognizing P_3-structure: A Switching Approach

Ryan B. Hayward
hayward@cs.uleth.ca

Report TR-CS-01-94

revised June 1995